ARMY RESEARCH LABORATORY

The BRL-CAD Package: An Overview

by Phillip C. Dykstra

ARL-RP-432 **April 2013**

A reprint from the *Fourth USENIX Computer Graphics Workshop*, Cambridge, MA, 9 October 1987.

NOTICES

Disclaimers

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5068

ARL-RP-432 **April 2013**

The BRL-CAD Package: An Overview

Phillip C. Dykstra
Survivability/Lethality Analysis Directorate, ARL

A reprint from the *Fourth USENIX Computer Graphics Workshop*, Cambridge, MA, 9 October 1987.

REPORT DOCUMENTATION PAGE					*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**					
1. REPORT DATE *(DD-MM-YYYY)* April 2013	**2. REPORT TYPE** Reprint				**3. DATES COVERED (From - To)** October 1987
4. TITLE AND SUBTITLE The BRL-CAD Package: An Overview					**5a. CONTRACT NUMBER**
					5b. GRANT NUMBER
					5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Phillip C. Dykstra					**5d. PROJECT NUMBER**
					5e. TASK NUMBER
					5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-SLB-S Aberdeen Proving Ground, MD 21005-5068					**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-RP-432
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)					**10. SPONSOR/MONITOR'S ACRONYM(S)**
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES A reprint from the *Fourth USENIX Computer Graphics Workshop*, Cambridge, MA, 9 October 1987.					
14. ABSTRACT The major components of the Ballistics Research Laboratory Computer-Aided Design (BRL-CAD) Package are reviewed. The BRL-CAD Package is a combinatorial solid geometry based modeling system which includes an interactive model editor, a ray tracing library, a generic framebuffer library, and a large collection of related tools. An object-oriented ray tracing library provides the primary method of model interrogation. A whole family of engineering analysis applications based on the ray tracing paradigm has been built, including traditional renderers and predictive radar models. A generic framebuffer library interface with transparent networking capability provides hardware independent access to any display device from any host. Several categories of software tools for image display, manipulation, and analysis are discussed. Some general user interface issues are mentioned. This paper emphasizes the reasons which led to the system as is exists today, and comments on some of its various strengths and weaknesses.					
15. SUBJECT TERMS NURBS BSpline, raytracing, CSG, BRL-CAD					
16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Clifford W. Yapp
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	14	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-1382

The BRL CAD Package
An Overview

Phillip C. Dykstra

Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

ABSTRACT

The major components of the BRL CAD Package are reviewed. The BRL CAD Package is a combinatorial solid geometry (CSG) based modeling system which includes an interactive model editor, a ray tracing library, a generic framebuffer library, and a large collection of related tools.

An object-oriented ray tracing library provides the primary method of model interrogation. A whole family of engineering analysis applications based on the ray tracing paradigm has been built, including traditional renderers, and predictive radar models. A generic framebuffer library interface with transparent networking capability provides hardware independent access to any display device from any host. Several categories of software tools for image display, manipulation, and analysis are discussed. Some general user interface issues are mentioned.

This paper emphasizes the reasons which led to the system as is exists today, and comments on some of its various strengths and weaknesses.

1. Introduction

The Ballistic Research Laboratory CAD Package is a large body of software consisting mainly of 1) a solid model editor (MGED), 2) a ray tracing library for model interrogation (librt), 3) a generic framebuffer library with full network display capability (libfb), and 4) a large collection of software tools for framebuffer and image manipulation and analysis. Parts of this system have roots in work done over two decades ago, most notably the solid modeling, and the ray tracing. Recently this software has been through a new generation of growth. It is now distributed free of charge to many sites around the world on a non-redistribution basis.

As with many large systems, parts of it were the result of years of evolution, with many band-aids, hacks, and "backward compatibility" requirements along the way. The work that one needed to accomplish today was often more influential than any carefully made plans. Most of this history is known only to those who watched it happen.

This paper provides a brief overview of the major components of the BRL CAD system. It will attempt to explain how and why many parts of it are the way they are. Finally, it will entertain the question of what is good and bad about it, and how the various decisions that were made have or have not worked.

2. Solid Modeling - MGED

The BRL has been building solid models of vehicles and other objects for over twenty years. These models are analysed for various physical properties (such as center of mass, moments of inertia), vulnerability, and more recently for optical, radar, and IR signatures.

This work began in the early 1960s when BRL had the Mathematical Applications Group Inc. (MAGI) develop a method of geometric description for military vehicles.[1] The method decided upon was Combinatorial Solid Geometry (CSG). This is a system where various geometric solids (boxes, cones, ellipsoids, tori, etc.) are combined using boolean operations (union, intersection, and subtraction). CSG represents one of the two major classes of modeling, the other being surface or boundary representations (B-reps). A key reason for the selection of CSG modeling is that it is "true to reality." Physical objects are solids, not just surfaces. If an object has been constructed with CSG, one is at least assured of its physical possibility.

For several years, models were constructed on large sets of punch cards. One or more cards would contain the parameters for a particular solid; other cards would describe the boolean relationships between solids. This system was not hierachical, all solids and combinations existed at one level. Ray tracing was used to analyse these models, but the only images of these models ever produced were crude plotter drawn wireframes.

A new generation of modeling tools emerged in 1979-1980. A system was built which allowed these models to be interactively displayed and edited on vector display devices. The success of these early efforts, coupled with the failure to find commercial tools of sufficient power, led to the development of the MGED model editor. The MGED editor is written in C and has been run on a large variety of machines. An object oriented interface to a set of display managers allows many different display devices to be supported. The types of primatives supported include: arbitrary boxes of up to eight verticies, ellipsoids, truncated general cones, tori, polygonal solids, and solids constructed of B-spline surfaces.[2]

The CSG representation is a natural form for our most common method of model interrogation - ray tracing. There are some methods of analysis however for which a surface facet representation of a model is the desired form. Work is currently under way on the facetization of CSG models, in order to support the needs of such codes. Future work is also planned in automatic mesh generation for similar reasons. These two capabilities will further ease the barrier between model representation, and model analysis.

For a much more comprehensive coverage of solid modeling, with MGED as a case study, see Muuss.[3]

3. Model Analysis - Ray Tracing

Ray tracing is a method of point sampling a geometric model by mathematically intersecting lines with objects in the model. At each intersection point various properties of the model can be determined: where did it intersect, what is the surface normal and curvature at that point, what part of the model was hit, what are the material properties at that point, etc. The computer graphics community often cites the origins of ray tracing with Kay's 1979 thesis,[4] or Whitted's paper of 1980.[5] However, the use of ray tracing as a method of geometric model interrogation has its origin in a BRL contract with MAGI, the initial results of which were published in 1967.[1] More details on the origins of ray tracing can be found in Muuss.[6] For an overview of the method itself, see Rogers.[7]

Ray tracing is the primary method used by BRL for model interrogation. Many people in the computer graphics community dislike ray tracing, primarily due to its notoriously high computational expense compared to other rendering techniques. But there are several key reasons why BRL uses it: 1) We are primarily concerned with doing an engineering analysis of the model, not just making pretty pictures of it, this objective is what led us to CSG models to begin with. 2) When CSG models are used, ray tracing is the most common method for evaluating the boolean expressions, 3) Firing a ray at a model is very much like firing a projectile (or light) at it, and is thus a natural method for vulnerability and signature analysis.

The ability to intersect rays with a model is common to all of the analysis tools, whether one is rendering a picture of the model or computing a moment of inertia. For this reason, the code which knows how to efficiently trace rays through a CSG model has been put in a library, librt. An application linked to this library has complete control over which rays are fired, how much information is computed at the intersection points, and what is done with the returned information. This library level separation of ray tracing and analysis has proven to be an extremely good one.

Other splits between ray tracing and analysis have been made or proposed. Some systems trace the entire model, placing the results into an intermediate file. There are two problems with this: the analysis code can not influence the ray trace (for example, by deciding when to reflect or when to fire extra rays in an area), and the volume of data generated is extremely large, often filling an entire large disk drive. The split could also be implemented by passing messages between separate processes via a remote procedure call, or a stream mechanism such as a UNIX pipe. The amount of overhead involved with either of these methods is typically of the same order of magnitude as the work involved in tracing a single ray. This approach is thus felt to be impractical.

Two ray tracing programs which use librt are provided in the CAD package: RT and LGT. LGT is an optical rendering program with a *curses* based screen oriented user interface. RT also provides rendered images with command line arguments, but is itself the front end for several applications including a radar model. RT also has the ability to read scripts of commands which can control the computation of a sequence of frames, and the orientations and properties of materials in each frame of an animation.

Future work with the ray tracer includes extending the classes of traceable objects, further efficiency improvements, and its extension to handle a broader class of physical phenomena. The latter goal includes multiple spectral point sampling (instead of just Red Green Blue) to account for dispersion and complex spectra, divergence factors (for the concentration and diffusion of light), and polarization effects.

4. The Framebuffer Library

The framebuffer library (libfb) provides a device independent interface to a raster display. A program compiled with this library can access many different display types, including those on other machines on the network. The most important routines are summarized below.

libfb routines	
fb_open(device,width,height)	open the device
fb_close(fbp)	close the device
fb_read(fbp,x,y,buf,count)	read count pixels at x,y
fb_write(fbp,x,y,buf,count)	write count pixels at x,y
fb_clear(fbp,color)	clear to an optional color
fb_rmap(fbp,colormap)	read a colormap
fb_wmap(fbp,colormap)	write a colormap
fb_window(fbp,x,y)	place x,y at center
fb_zoom(fbp,xzoom,yzoom)	pixel replicate zoom
fb_getwidth(fbp)	actual device width in pixels
fb_getheight(fbp)	actual device height
fb_cursor(fbp,mode,x,y)	cursor in image coords
fb_scursor(fbp,mode,x,y)	cursor in screen coords
fb_log(format,arg,...)	user replaceable error logger

The coordinate system for x,y specifications is first quadrant. While we went round and round about first vs. fourth quadrant with arguments akin to "which end of the egg first", the decision for first quadrant resulted primarily because that is the same ordering as our image files (.pix files, see below). The image files themselves were ordered that way because Utah's RLE files are first quadrant. If reads and writes extend beyond the end of a scanline, they wrap in first quadrant fashion.

The pixels passed to and from the library are simply arrays of bytes interpreted as RGBRGB.... While we used to define a pixel structure with red, green, and blue elements, this was changed to a typedef'd array of three unsigned chars. This was important in order to avoid structure padding. The Cray computers for example would have used eight bytes per pixel with the old format. Unfortunately, one does run into some compiler touchiness when using pointers to typedefs which are themselves arrays!

The display to be used is selected by a command line argument, an environment variable FB_FILE, or a default for the system the code is running on. The format is [host:]/dev/*device_name*[#], or simply "filename". The /dev/ part is used to identify a display device. The *device_name* need not correspond to entries in /dev, it is just that if the /dev prefix is not given a file pathname is assumed. If a hostname is given, a network connection is opened to the framebuffer library daemon (rfbd) on that machine. The remaining part of the string is passed to that host for the open (this generalizes the open to allow multiple "hops" in order to get to a host). Currently supported displays include the Adage Ikonas, Silicon Graphics Iris, black and white and color Sun workstations, and AT&T 5620 terminals. There is also a debug interface, and a disk file interface.

A set of buffered I/O routines is also provided. In this interface a "band" of scanlines is kept in memory and the appropriate pre-reads and flushing is done. While this interface can speed up single pixel reads and writes, it does not make the drawing of vertical lines any easier, since such a line would run through several bands. In practice, very few of our programs use buffered I/O. Most programs keep their own scanline buffers and do unbuffered scanline size reads and writes. Some thought has been given toward allowing the selection of the memory buffering mode at run time, perhaps keyed on a device name parameter. This would permit the user to control the trade off between speed and interactive output. The ability to make such a decision becomes particularly important when one is using a remote display.

libfb buffered I/O	
fb_ioinit(fbp)	set up a memory buffer
fb_seek(fbp,x,y)	move to an x,y location
fb_tell(fbp,xp,yp)	gives the current location
fb_rpixel(fbp,pixelp)	read a pixel and bump location
fb_wpixel(fbp,pixelp)	write and bump current location
fb_flush(fbp)	bring display up to date

The framebuffer library owes much of its current form to its history. One of the first true framebuffers purchased by BRL was an Ikonas (now Adage RDS-3000), in 1981. This device runs as either a 512x512 or 1024x1024 display with 24 bit pixels. It has three 256 entry 30-bit (10 bits per DAC) colormaps, hardware pan and zoom, and hardware cursor support. Michael Muuss of BRL wrote our first library for that device (libik).

Later, a Raster Technologies One/180 framebuffer was acquired and a libik like interface was created for it. As other devices followed, libfb was born. At first there was a switch in every library routine for every display device. Later it was reworked to have an object oriented interface: opening a device fills in a function switch table with that display's routines, and a "framebuffer pointer" was returned to that structure. Most of the framebuffer routines became macros which vector directly out to the device dependent code.

Finally, the machine which had our nice displays on it (a VAX 11/780) was also one of our slowest. To make this less of an issue, a libfb look alike was put together one evening which passed all library calls and returns across a network connection to a daemon that made calls to a "real" libfb. This was facilitated by the Package Protocol[8] (PKG) which allows messages to be exchanged, both synchronously and asynchronously, across a TCP connection (this protocol had originally been developed to make a remote MGED display possible, but later found uses in command and control experiments, etc.). The remote framebuffer code was merged into libfb during its object-oriented restructuring, so that one need only link with a single library to get both local and remote display capability.

Starting with the Ikonas in some sense spoiled us. It gave us full color pixels, colormaps, cursors, and pan and zoom. These features were incorporated into the generic framebuffer model used in our library. This makes fitting devices like the Sun workstations into our library quite trying, but this difficulty is more the result of things that workstations like the Sun can't do than it is a design problem with our library. On the other hand, the Ikonas also left us with programs that have to open the device in one of two "modes", either high or low resolution. To make matters worse, it does not allow the current display mode to be read back from the hardware. Therefore, the open must set the Ikonas to a known state. As a result, every framebuffer program, even those which have little to do with display size (such as those which read or write colormaps), carries around a "hires" flag so the device can be opened in the proper "mode."

One commonly asked question is whether X Windows will make the BRL framebuffer library unnecessary. X currently cannot support 24-bit color images, nor does it provide a powerful enough interface for controlling many framebuffer operations (e.g. colormaps, pan and zoom). If these deficiencies are overcome than X may prove to be a suitable replacement for the framebuffer library. In the near term, an X based module implementing a subset of the framebuffer library functions will likely be developed.

5. The Software Tools

A large number of simple tools for manipulating images and framebuffers are provided in the CAD package. They have been written in the traditional UNIX Software Tools fashion: each performs a simple basic function, with a minimum of back talk, and is intended to be hooked together with other tools to achieve an overall goal. A fair amount of effort has gone into making a standard interface to the tools. All tools provide a usage message if executed with no arguments (often after checking for a tty on stdin or stdout when it expects binary data), and common collection of flags is defined for all of the tools.

The use of software tools for computer graphics is not new. Recent systems advocating this tools based approach include those of Duff[9] and Peterson.[10] The BRL CAD Package has proven to be extremely flexible as as result of this approach. Generally, a new tool is added whenever the existing ones are found to be inadequate. Success can be claimed if one can easily achieve day-to-day tasks without having to write specialized programs.

5.1. File and Image Formats

Several kinds of files are read and generated by programs in the CAD package. These include model databases in a binary form (with a typical filename extension of .g), portable ASCII versions of those (.asc), and University of Utah Run Length Encoded (RLE) images (.rle). By far the most common image format for the tools however is either eight bit per pixel black and white (.bw) or 24-bit per pixel color (.pix). The files have the simplest format imaginable: there is no header at all, and pixels run in first quadrant order - lower left corner, across the scan lines, bottom scan line first, up through the top scan line. The values in the bytes are viewed an intensities from 0 (off), through 255 (full on). The color (.pix) files are in RGBRGB... order. Note that while we use the University of Utah RLE format, we view it simply as a means of image compression, unlike Utah which actually manipulates RLE files directly in their Raster Toolkit.[10]

The use of a simplistic headerless image format is perhaps the most debatable decision we made. It's primary advantage comes when piping several tools together. Each program is simply handed data. It doesn't have to know "how" to read it; there is no header to discard, or harder still, it doesn't have to do the "right thing" with the header information. Doing the "right thing" is extremely complicated if the header contains very much information. We have also avoided the N^2 problem of format conversion by converting all other formats into and out of this simple one.

Having "raw" headerless data has its price however. It is difficult to tell whether a given image is color or not, what its dimensions are, etc. File naming conventions (.bw or .pix) solve the first; "standard sizes" of 512x512 or 1024x1024 (hires) help alleviate the second (recall that these came from the Ikonas framebuffer). Note that usually only the scanline length needs to be

known, the number of lines can then be found by the file size. Many algorithms simply run until all of the data is gone, and some don't even care about scanlines at all.

5.2. Format Conversion

Several other image formats are accommodated by "filters" that convert one into the other. A selection of these is listed in the table. In all of the tables given the reverse conversion is omitted, e.g. there is also a pix-rle for converting color images into RLE format. Also, only the color (pix) version of a tool has been shown while most have black and white (bw) equivalents. Most of the tools listed also allow a wide variety of options. The color to black and white converter for example (pix-bw), allows either equal, NTSC, or "typical" CRT weighting to be applied. It also allows arbitrary weights to be given for selecting or mixing of the color planes in any way desired.

Selected Format Conversion Tools	
g2asc	model database to portable ascii form
bw-pix	black and white to color image
bw3-pix	three black and whites to color RGB
rle-pix	Utah's RLE format to color image
ap-pix	Applicon Ink-Jet to color image
sun-pix	Sun bitmap to color or black and white
mac-pix	MacIntosh MacPaint bitmaps to color

5.3. Framebuffer Tools

We have chosen to do most of the image manipulation and processing either on data streams, or on disk files. This was done in order to separate the notion of a device from image handling. A common beginning or end of a processing pipeline is to get or put an image into or from a framebuffer. Framebuffers do allow one to manipulate images in many useful ways however, so some device independent tools are provided for that. These include tools to allow changing colormaps, panning and zooming through an image, labeling, etc. Where tools require the user to move a cursor or the image, both EMACS and VI style commands are accepted by all programs.

Selected Framebuffer Tools	
fb-pix	framebuffer to color image
fb-bw	framebuffer to black and white
fb-cmap	read a framebuffer colormap
fbcmap	can load several "standard" colormaps
fbclear	clear to an optional RGB color
fbgamma	load or apply gamma correcting colormaps
fbzoom	general zoom and pan routine
fbpoint	select pixel coordinates
fblabel	put a label on an image
fbcolor	a color selecting tool
fbscanplot	scanline RGB intensity plotter
fbanim	a "postage-stamp" animator
fbcmrot	a colormap rotator
fbed	a framebuffer image editor

5.4. Image Manipulation

A collection of tools for image manipulation are provided. These can generate statistics, histograms, extract parts of an image, rotate, scale, and filter them, etc. Some of these are listed in the table.

Selected Image Tools	
pixstat	statistics - min, max, mean, etc.
pixhist	histogram
pixhist3d	RGB color space cube histogram
pixfilter	apply selected 3x3 filters
pixrect	extract a rectangle
pixrot	rotate, reverse, or invert
pixscale	scale up or down
pixdiff	compare two images
pixmerge	merge two/three images
pixtile	mosaic images together
gencolor	source a byte pattern
bwmod	apply expressions to each byte

6. User Interface

Using software tools effectively comes with experience. The BRL CAD Package has tried to ease the difficulty of learning a new set of tools by using a common set of flags and common tool naming conventions throughout the package. The "user interface" is ultimately the Unix shell, and its conventions for establishing pipes, passing arguments to programs, etc. A shell with history recall and editing, such as the *tcsh*, is almost a necessity when constructing complicated command line pipes.

Constructing complex interconnections between processing tools from the command line is sometimes difficult. One limitation is the single input single output notion of a Unix pipe. Image manipulation often calls for three or more channels of data. The most common solution to this problem is the use of intermediate files. Other approaches include extensions to the *tee* program, or a special tool such as *chan*[11] which demutiplexes a stream, feeds each channel to a different program, and remultiplexes the results.

Recently several systems have been developed to facilitate the coupling of dataflow oriented tools. Stephen Willson of NRTC has developed what he calls a Layered User Interface.[12] This is a set of tools that provides generic buttons and sliders which can pass values on as tool arguments. Several of the BRL CAD tools have been used in this environment. Dave Tristram of NASA Ames has put together a system called Flowtools[13] which allows the connections between tools to be specified with a dataflow like language, including inputs from sliders, etc. Both of these systems allow complex custom applications to be put together without writing any code.

7. Conclusions

The BRL CAD Package is a Unix based system which provides a CSG solid model editor, a ray tracing library for model interrogation, a generic framebuffer library with network display capability, and a large collection of software tools. The library level interface to the ray tracer has allowed a large collection of model analysis tools to be incorporated into the system. The generic network capable framebuffer library has proven to be of tremendous day to day importance.

The package provides a flexable set of software tools for image manipulation. The image formats are extremely simplistic, something which has proven to have both good and bad characteristics. Approaches to providing higher level interfaces to tools of this form have been indicated.

1. MAGI Inc, *A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles,* MAGI Report 6701, AD847576 (August 1967).
2. P. R. Stay, "The Definition and Raytracing of B-spline Objects in a Combinatorial Solid Geometric Modeling System," *USENIX: Proceeding of the Fourth Computer Graphics Workshop* (Oct 1987).
3. M. J. Muuss, "Understanding the Preparation and Analysis of Solid Models," in *Techniques*

for Computer Graphics, ed. D. A. Rogers, R. A. Earnshaw, Springer-Verlag (1987).

4. D. S. Kay, *Transparency, Refraction, and Ray Tracing for Computer Synthesized Images*, Cornell Univ (Jan 1979).
5. J. T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM* **23**(6), pp. 343-349 (June 1980).
6. M. J. Muuss, "RT and REMRT - Shared Memory Parallel and Network Distributed Ray-Tracing Programs," *USENIX: Proceeding of the Fourth Computer Graphics Workshop* (Oct 1987).
7. D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, New York (1985).
8. M. J. Muuss, P. Dykstra, K. Applin, G. Moss, E. Davisson, P. Stay, C. Kennedy, *Ballistic Research Laboratory CAD Package, Release 1.21*, BRL Internal Publication (June 1987).
9. Tom Duff, "Compositing 3-D Rendered Images," *Computer Graphics 19(2):41 (Proceedings of SIGGRAPH 85)* (July, 1985).
10. J. W. Peterson, R. G. Bogart, and S. W. Thomas, "The Utah Raster Toolkit," *USENIX: Proceeding of the Third Computer Graphics Workshop* (1986).
11. R. F. Moore, *CARL Startup Kit*, Computer Audio Research Laboratory, UCSD (1985).
12. S. Willson, "The Layered User Interface," *IRIS Universe* (To appear, Fall 1987).
13. David Tristram, "FlowTools: Dataflow Graphics Under Unix," *to appear, IEEE Conference on Workstations*, NASA Ames Research Center.

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1 (PDF)	GOVT PRINTG OFC A MALHOTRA 732 N CAPITOL ST NW WASHINGTON DC 20401
1 (PDF)	USARL RDRL SLE R FLORES WSMR NM 88002-5513
	ABERDEEN PROVING GROUND
1 (HC)	DIR US ARMY EVALUATION CTR HQ TEAE SV P A THOMPSON 2202 ABERDEEN BLVD 2ND FL APG MD 21005-5001
3 (2 HC 1 PDF)	DIR USARL RDRL SL J BEILFUSS P TANENBAUM RDRL SLB A M PERRY (PDF only)
7 (6 HC 1 PDF)	RDRL SLB G KUCINSKI RDRL SLB S S SNEAD (5 CPS) C YAPP (1 PDF)
4 (HC)	QUANTUM RSRCH INTRNTL C HORTON STE 203 2014 TOLLGATE RD BEL AIR MD 21015

INTENTIONALLY LEFT BLANK.